Développement d'une plate-forme de supervision d'un réseau de capteurs

Arthur Garnier, Emmanuel Nataf

Développement d'une plate-forme de supervision d'un réseau de capteurs

Arthur Garnier

Emmanuel Nataf

MADYNES : Management of Dynamics Networks. Equipe-projet INRIA
L.O.R.I.A, Campus Scientifique, 615 Rue du Jardin Botanique, 54506 Vandœuvre-lès-Nancy

Table des matières

Première partie

Introduction

Étudiant en deuxième année de DUT informatique j'ai effectué un stage au sein de l'équipe Madynes au LORIA, ce stage portait sur du développement JEE et SGBD.

L'équipe Madynes, effectue actuellement des recherches sur les réseaux de capteurs, mon travail a été de proposer et réaliser une application web capable de représenter de façon simple et ergonomique ces données, l'application devant rester extensible et adaptative aux données qui lui sont fournies par les capteurs. Ce stage a été l'opportunité pour moi d'appréhender au mieux le monde professionnel de l'informatique.

Au delà des compétences acquises, ce stage m'a permis de découvrir le monde de la recherche en informatique et surtout l'importance de la veille technologique, afin d'exploiter au mieux les outils mis à notre disposition.

Je commencerai par présenter l'état d'avancement du projet lors de mon arrivée ainsi que mes objectifs, puis par quels moyens j'ai accompli ces derniers.

Deuxième partie

Présentation

Chapitre 1

L'entreprise

Le Loria, laboratoire lorrain de recherche en informatique et ses applications, a été créé en 1997, il a, depuis sa création, pour mission la recherche fondamentale et appliquée en sciences informatiques. Le laboratoire est composé de vingt-huit équipes divisées en cinq départements. L'équipe Madynes dans laquelle je me trouve fait partie du département *Réseaux, Systèmes & Services*, elle est composée de 11 personnes. Un des thèmes de recherche actuel de l'équipe étant les réseaux de capteurs, le besoin de stocker et visualiser les données issues de ces derniers de manière intuitive s'est posé.

Chapitre 2

Le travail réalisé

2.1 État des lieux

Lors de mon arrivée au LORIA, j'ai pu profiter d'un travail déjà avancé à propos desdits capteurs, en effet le réseau de capteurs était déjà fonctionnel, avec un nœud central, appelé *sink*, ce sink étant capable de retransmettre les données reçues sous forme de chaîne de caractère ASCII. Mon travail ayant été réalisé en collaboration avec deux étudiants en deuxième année à Telecom Nancy dans le cadre de leur PIDR [1], ces deux étudiants avaient déjà réfléchi à une structure, pour la base de données et pour l'application, de ce fait une partie du modèle était prête.

2.2 Objectif

L'objectif principal est la création d'une application web permettant l'affichage de données exploitables. Dans le cadre d'une collaboration avec une entreprise, l'application devait être orienté vers l'exploitation de données de température, mais tout en restant modulable et extensible. Les capteurs, dans le cadre de cette collaboration, serviraient à comparer différentes technologies de chauffage en mesurant les températures dans un même local.

1. Projet Interdisciplinaire de Découverte à la Recherche

Troisième partie

Le développement

Chapitre 3

L'architecture et les capteurs

3.1 Fonctionnement global du réseau de capteurs

3.1.1 Qu'est ce qu'un capteur ?

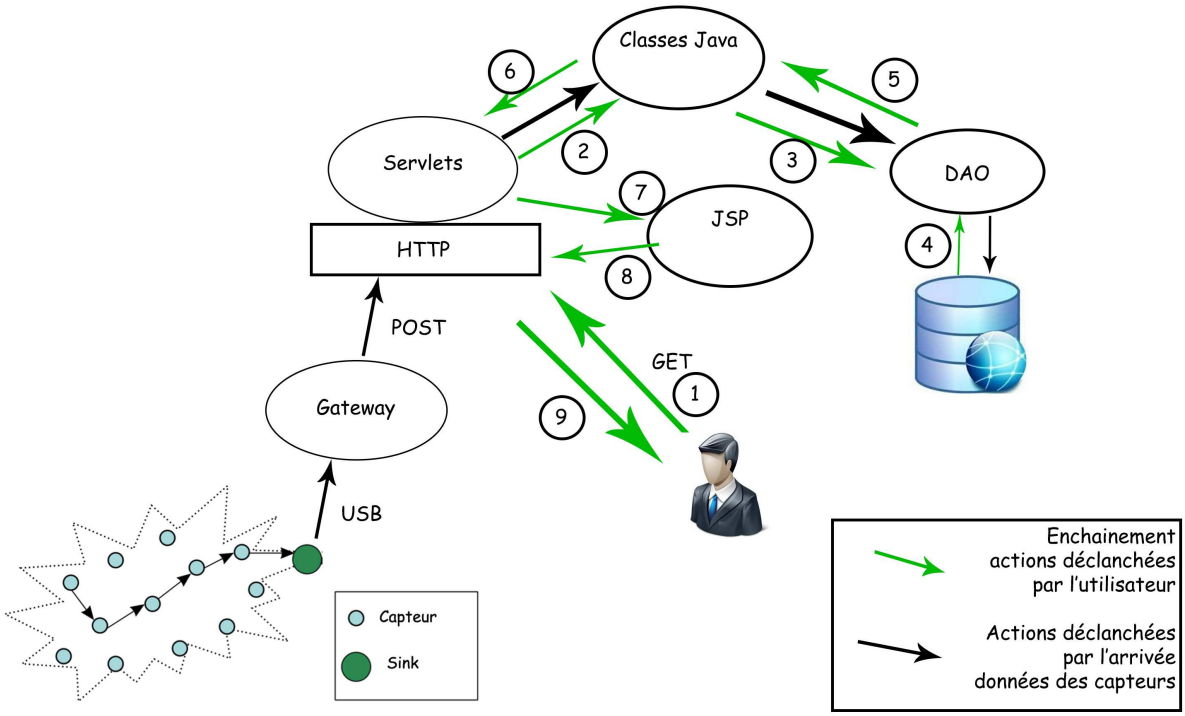
Un capteur est un dispositif permettant de mesurer des grandeurs physiques comme la consommation électrique, la température, l'humidité, ... Ici nous appelons plus généralement le capteur l'ensemble du système capable de mesurer la donnée, c'est à dire le capteur de la mesure en lui même, le CPU [1], le module de communication, l'alimentation et la mémoire.

3.1.2 Un réseau de capteurs

Ces capteurs font partis de ce que l'on appelle l'internet des objets, en effet ils sont capables de créer leur propre réseau de manière intelligente, ce sont des « smart-objects », et de transmettre leur données vers l'Internet. En réalité il existe deux type de composant dans un réseau de capteurs, les capteurs, et le *sink*, ce dernier n'a pas la charge de mesurer mais uniquement de récupérer les données de tous les autres capteurs du réseau. Les capteurs eux, réalisent des mesures à intervalle régulier et les envoient au sink, soit de manière directe soit en étant relayées par les autres capteurs, qui vont répéter l'information jusqu'au sink si la porté du capteur initial ne permet pas de l'atteindre directement.

Note : Tous les capteurs ne sont pas capable de relayer l'information des autres, cette description est valable pour le cas étudié.

FIGURE 3.1 – Schéma de l'architecture de l'application



3.2 Architecture de l'application

3.2.1 Les données des capteurs

Le schéma ci-dessus représente la structure générale de l'application ainsi que le parcours des informations dans celle-ci en fonction des utilisations. Je commencerai par détailler le parcours d'une information provenant d'un capteur, elle est d'abord envoyée du capteur jusqu'au sink, ce sink est branché en USB à un ordinateur (ou un raspberry Pi par exemple) où est lancé la gateway. La gateway est un programme qui écoute sur le port USB où est branché le sink, ce dernier envoie des chaînes de caractères du type « 3460 223 1 62504 21941 8 742 1 2097 253 3975 2071 0 180 222 652 », le seul rôle de la gateway étant de retransmettre ces informations au serveur qui traitera ces données.

Dans notre cas, la chaîne est renvoyée vers un serveur web JEE (Glassfish), donc vers une servlet qui a pour rôle d'interfacer l'internet avec le programme interne en fonction des actions effectuées, c'est un contrôleur. Donc la servlet reçoit la chaîne de caractère, elle est transmise vers des classes Java qui vont s'occuper du traitement de celle-ci pour en extraire les informations. Ces classes ont des stratégies à appliquer en fonction du numéro de la colonne de la chaîne de caractère

1. Central Processing Unit ou Processeur

afin de les transformer en valeur réelle. Par exemple ici :

1. Timestamp
2. Id du capteur
3. Numero de séquence
4. Nombre de sauts
5. Id du parent
6. Taux de transmission vers le parent
7. Métrique
8. Nombre de voisin
9. Intervalle de temps des messages du protocole RPL
10. Batterie restante
11. Batterie restante (suite)
12. Tension
13. Indicateur de batterie
14. Capteur de lumière 1
15. Capteur de lumière 2
16. Capteur de température
17. Capteur d'humidité

Pour chaque valeur extraite de cette chaîne, un objet *Data* est créé, et, transmis au DAO [2] qui va le stocker dans la base. Le DAO est une interface entre le programme et la base de donnée, ce modèle permet de ne pas se soucier comment les données sont stockées, de cette façon un changement de mode de stockage (du SGBD à XML par exemple), ne remet pas en cause le fonctionnement du reste de l'application, seul le DAO devra être modifié.

3.2.2 L'interaction avec l'utilisateur

Sur le schéma, on peut également voir les interactions entre les parties de l'application, représentées par les flèches vertes. Lorsqu'un utilisateur interagit avec l'application, il passe toujours à travers les servlets via le protocole HTTP. Les servlets, interfaces entre l'utilisateur et le reste de l'application, sont des contrôleurs, en fonction de la page appelée une servlet différente peut-être appelée, puis celle-ci en fonction des paramètres envoyés par le navigateur va interagir avec certaines classes de l'application (étape 2). Ces classes réalisent des traitements, ils peuvent aller de la simple conversion à la récupération de données dans la base de données

2. Data Access Object

via le DAO (étapes 3, 4 et 5) et retourne le résultat à la servlet (étape 6). La servlet redirige vers la page JSP correspondante avec les éventuelles données traitées (étape 7) puis la page est renvoyée vers l'utilisateur via HTTP (étape 8 et 9).

Chapitre 4

Les technologies employées

4.1 JSP/JSTL

Le début de l'application développée par les deux étudiants de Telecom Nancy étant basé sur Java EE, ma première mission a été d'apprendre cette variante du langage Java jusqu'alors jamais étudiée. Bien que basée sur Java SE au niveau du code dans des servlets [1] et plus ou moins équivalent dans le modèle. En revanche concernant la vue, un méta-langage complètement différent de Java apparaît : JSP. C'est une technologie qui permet d'insérer du code Java dans du HTML qui sera interprété au moment de la compilation par le serveur JEE. Néanmoins l'insertion directe de code Java dans le code HTML étant déconseillé et assez peu lisible, JSP fait intervenir des balises permettant d'accéder aux variables des paramètres passés à la page web.

JSP peut être combiné avec JSTL, qui est une extension de JSP qui permet d'effectuer des actions un peu plus développées que l'accès aux variables, on peut par exemple réaliser des boucles conditionnelles, ou appeler des fonctions sur des chaînes de caractère, ... Il faut tout de même savoir que JSP et JSTL restent du Java, mais « masqué » derrière des macros.

Voici une comparaison de JSP/JSTL face à du Java simple :

```
1  <c:forEach var="item" items="${liste}" >
2          <c:out value="${item}" />
3 </c:forEach>
```

1. Classe Java faisant office de contrôleur dans le modèle MVC

```
1  List<Integer> list =
2   (ArrayList<Integer>)request.getAttribute("liste");
3         for(int i = 0; i < list.size();i++){
4                 out.println(list.get(i));
5         }
```

On remarque que le premier code est plus abstrait que du Java. Les pages JSP étant destiné à un designer par exemple, cette technologie est plus abordable que du Java. Dans le précédent exemple, on peut voir une balise avec le préfixe « c », qui est le préfixe d'appel à la bibliothèque *Core* de JSTL qui fournit les actions de base, comme le parcours de liste (ligne 1 : `forEach`), l'encodage d'URL, l'affichage de variables (ligne 2 : out)... JSTL est composé, dans sa version 1.1, de 5 bibliothèques permettant de réaliser des actions orientées sur certains domaines.

Les autres bibliothèques sont :

— Format : les opérations sur les dates
— XML : le traitement de balises XML
— SQL : les interactions avec les bases de données
— Functions : les manipulations des chaînes de caractères

4.2 EJB

Une autre notion a dû être assimilée, celle des EJB [2], qui forme une architecture de composants logiciel pour représenter des données, pour proposer des services et pouvant évoluer dans un contexte transactionnel. Une propriété importante est que les EJB peuvent être persistant, c'est à dire qu'ils peuvent être stockés sur un support physique en vue d'une réutilisation. Une autre propriété est qu'ils ont la possibilité ou non de garder leur état entre les appels (Stateless/Stateful). L'ensemble de ces caractéristiques en font l'outil idéal pour l'application, en effet, nous avons le besoin de stocker les EJB dans une base de données avec leurs caractéristiques.

4.3 Javascript

Pour les interfaces utilisateurs nous avons souhaité de l'interactivité, nous avons donc opté pour JavaScript. C'est un langage de programmation de script qui s'exécute généralement dans un navigateur, il est compatible avec les plus répandus, sous réserve d'une version relativement récente. Il a également l'avantage de s'exécuter côté client et donc de limiter la charge du serveur. De plus, il existe

2. Enterprise JavaBean

beaucoup de frameworks et de librairies pour Javascript, que nous avons largement utilisé pour le développement de l'application, et plus particulièrement trois :

1. JQuery [3] : Un framework très utile pour les interactions Javascript/HTML
2. Highcharts [4] : Une librairie pour réaliser des graphiques interactifs
3. KineticJS [5] : Un framework permettant de représenter des formes et de les manipuler grâce aux canvas HTML5

3. https://jquery.com/
4. http://www.highcharts.com/
5. http://kineticjs.com/

Chapitre 5

Développement itératif

Chaque nouvelle fonction à implémenter dans l'application s'est déroulée en plusieurs étapes, toujours quasiment identiques, formant des cycles. C'est ce qu'on appelle un développement itératif. Il permet une gestion du calendrier plus adaptée car la progression du développement est quasiment linéaire. En cas de bugs, le temps d'identification du problème et de réparation est assez faible puisqu'il provient très certainement des dernières implémentations.

Notre développement s'est donc déroulé de cette façon pour chaque fonction :

1. Analyse des besoins
2. Conception
3. Codage
4. Tests

Cette façon de développer a permis d'avoir une vision, quasiment en temps réel, de l'avancement du travail et des choses qui pouvait manquer ou qui étaient à corriger. Comparé à un développement trop hâtif qui a l'inconvénient d'une mauvaise estimation des risques, d'un temps de correction des bugs trop long et qui ne permet pas d'avoir une visualisation de l'avancement du projet.

5.1 Premier cycle : Les graphiques

Les données transmisent par les capteurs étant des valeurs réelles datés, la première idée, presque naturelle, pour les représenter est un graphique en deux dimensions, le temps en abscisse et la valeur en ordonnée. La première période d'analyse des besoins a donc été plutôt courte, de même pour la conception, bien qu'une recherche d'une alternative efficace au javascript pour créer les graphiques ait été recherchée, Highcharts a finalement été désigné pour l'application.

Bien qu'une partie développée par les deux étudiants de Telecom permettait de voir les données en temps réel, nous avions besoin de voir les données même après

la fin de l'expérience. Une adaptation a donc dû être effectuée afin de récupérer les données correspondant à des paramètres donnés (Mesure, intervalle de temps, capteur).

5.1.1 Côté servlet

Le premier développement à faire était au niveau de la servlet, qui doit passer à la page du graphique (JSP) les données provenant de la base de données en fonction des paramètres qui étaient demandés (dans l'URL). Pour s'adapter à tous les types de paramétrage, il a fallu creer une requête SQL assez modulable.

Par exemple, si les paramètres de l'URL ressemblent à :

```
http://monsite.com/get?experiment=12&experiment=13
                    &label=temperature&dateDeb=2014/04/1815:20
```

Cela signifie que les données souhaitées sont :

— L'expérience 12

— L'expérience 13

— Uniquement les données dont le label est température

— A partir du 18/04/2014 à 15h20

De ce fait il faut générer la requête SQL suivante :

```
SELECT o FROM Data o where o.experiment.id = 12
    OR o.experiment.id = 13 AND o.label.label='temperature'
    AND o.timestamp>1397827200000
```

Tout d'abord on remarque que ce n'est pas exactement la syntaxe SQL, c'est en fait du JPQL[1], langage très proche du SQL, mais indépendant de la plateforme, ce qui signifie que peu importe le système SGBD utilisé, il fonctionnera de la même façon. De plus il permet une interaction avec des objets, comme on peut le voir, on peut accéder directement aux attributs des objets : *o.experiment.id* ; ce qui permet plus de flexibilité quand aux paramètres entrés. Par exemple, en SQL pour le label `temperature` nous aurions dû récupérer la clé primaire du label pour l'utiliser, ici on utilise directement le nom du label. On remarque également que la date est stockée dans la base sous forme de *timestamp* : une représentation de la date par le nombre de millisecondes depuis le 1er janvier 1970. Ceci permet des traitements plus simple pour les comparaison de dates car elles sont ramenées à des comparaison d'entier.

Pour transformer les paramètres en cette requête SQL plusieurs traitements de chaînes de caractères ont dû être effectués. Par exemple la transformation de la

1. Java Persistance Query Language

date saisie *2014/04/1815 :20* en *timestamp*, qui se fait grâce à la manipulation d'objet Java comme ceci :

```
1  public static Timestamp dateToTimestamp(String date)  {
2          DateFormat df;
3          Date d;
4          df = new SimpleDateFormat("MM/dd/yyyyHH:mm");
5          try {
6                  d = (Date) df.parse(date);
7          } catch (ParseException pe) {
8                  d = new Date();
9          }
10         Timestamp timestamp = new Timestamp(d.getTime());
11         return timestamp;
12         }
```

Ici, le format de date est défini à l'avance, il est donc indiqué à Java (ligne 4). Ce qui nous permet de transformer la chaîne de caractère en objet `Date` reconnu par Java, si ce n'est pas le cas la date actuelle est définie (ligne 8), et ensuite d'extraire le *timestamp* de cette date (ligne 10).

Une fois la requête générée, elle est transmise au DAO, les données sont ensuite retournées sous forme de liste d'objet `Data`. Or javascript ne supporte pas les objet Java, il faut donc structurer les données en JSON[2], qui est un format de données textuelles et générique à l'image du XML. En Java il est possible de « convertir » des objets Java en JSON, une méthode a donc été réalisée dans cette optique :

```
1  public static ArrayList<JsonObject> fromBDD(List<Data> list){
2  ArrayList<JsonObject> ret = new ArrayList<JsonObject>();
3  for (Data d : list) {
4   JsonObjectBuilder dataBuilder = Json.createObjectBuilder();
5   dataBuilder.add("mote", d.getMote().getIpv6());
6   dataBuilder.add("timestamp", d.getTimestamp());
7   dataBuilder.add(d.getLabel().getLabel(), d.getValueToStr());
8   dataBuilder.add("experiment_id", d.getExperiment().getId());
9   JsonObject jsonObject = dataBuilder.build();
10  ret.add(jsonObject);
11 }
12 return ret;
13 }
```

2. JavaScript Object Notation

Cette méthode prend en paramètre la liste de `Data` récupérée par le DAO, et pour chaque `Data` de cette liste (ligne 3), crée un objet Java spécialisé dans la construction d'Objet JSON (ligne 4), on lui ajoute les propriétés utiles (ligne 5 à 8) qui sont l'IP du capteur, la date et la valeur de la donnée et le numéro de l'expérience. Puis on crée un objet JSON (ligne 9) que l'on ajoute à une liste d'objets JSON qui sera retournée par cette méthode (ligne 10).

Les données, désormais prêtes, sont transmisent par la servlet à la page JSP destinée à contenir le graphique Highcharts.

5.1.2 Côté interface utilisateur : Graphiques

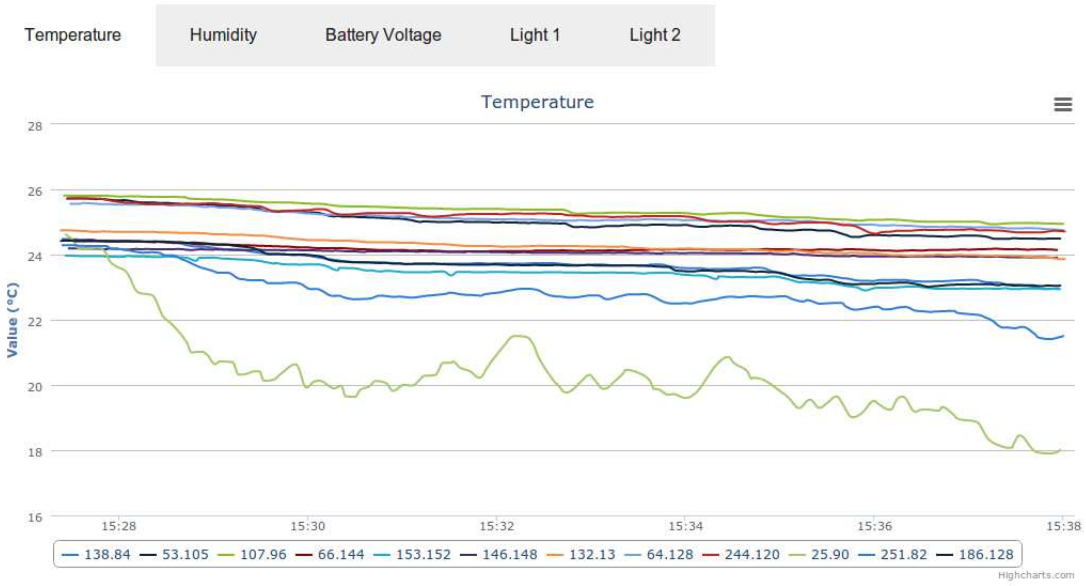
Le travail de développement au niveau d'Highcharts a surtout été d'adapter les données au mieux pour optimiser la vitesse de rendu. En effet, dans la première version développée, le temps de rendu était de 1.6 points par seconde, ce qui est très lent puisque le nombre de données provenant d'un réseau de capteurs peut-être de plusieurs milliers voire millions. Un travail d'approfondissement du fonctionnement d'Highcharts a donc été mené pour accélérer le rendu. Alors qu'une méthode était utilisée pour ajouter les points un par un au graphique, nous avons tenté d'ajouter directement tous les points dans les tableaux de la structure d'Highcharts. Grâce à cette méthode le rendu est supérieur à 10000 points par seconde, soit environ 6000 fois plus rapide.

Des améliorations afin de rendre le graphique le plus complet et précis possible ont été effectuées, comme par exemple l'ajout d'options permettant de zoomer sur une zone précise, d'activer les options d'export (JPG, PNG, SVG) et y ajouter le format CSV [3] qui n'est pas présent par défaut. Ces options d'export ont en revanche mené à un problème d'ordre technique, en effet par défaut les graphiques Highcharts exportent leurs valeurs sur son serveur publique (`export.highcharts.com`) pour les traiter et les transformer en image. Or, ce serveur est limité pour éviter la surcharge. Un grand nombre de données ne peuvent donc être exporté d'un seul coup. De ce fait une prise de contact avec un des créateur de Highcharts fut nécessaire afin de récupérer les sources du serveur d'export (disponible sur le dépôt Git du projet). Le serveur était disponible en JEE, donc facilement déployable sur notre serveur, il a ensuite suffit de modifier sa configuration pour autoriser une plus grosse charge. Les paramètres de la JVM [4] ont également été ajustés afin d'augmenter la mémoire allouée au serveur. Enfin nous avons indiqué à notre script Highcharts de diriger les demandes d'export vers notre serveur.

3. Comma-separated values
4. Java Virtual Machine

Au final nous obtenons un rendu du type de celui de la figure 5.1

FIGURE 5.1 – Aperçu du rendu d'un graphique



5.1.3 Côté interface utilisateur : La page de configuration

La page de configuration sert à saisir de manière simple et ergonomique les paramètres de l'URL qui, jusqu'à présent, devaient être saisis à la main. Cette page propose donc le choix des mesures à afficher, les dates de début et/ou de fin des données, celles en temps réel doivent également être affichées, et les différents capteurs (température, humidité, lumière, ...).

Pour afficher les mesures existantes, qui sont variables et donc qui ne peuvent figurer dans le code de la page web directement, il a fallu modifier la servlet de la page pour qu'elle envoie ces données (de la même manière que pour les données du graphique). Les boutons « checkbox » chargés de permettre la sélection de chaque mesure sont générés par du JSTL, créant un bouton par valeur dans la liste des mesures transmise par la servlet.

Afin de limiter la charge du serveur, JQuery a été utilisé pour générer le lien à l'instar d'un traitement de chaîne de caractères sur le serveur. JQuery est un framework javascript qui permet d'interagir avec les élément HTML de la page. Ici il a été utilisé pour déclencher une action au moment de l'utilisation du bouton *Submit*. Cette action consiste à analyser chaque élément sélectionner et générer l'URL appropriée qui sera transmise au serveur. Par exemple :

```
1 $("#live").each(function(i){
2     if($(this).is(':checked')){
3             condition+="live=true&";
```

```
4             }
5  })
```

Cet extrait de code vérifie si le bouton « live » est sélectionné (ligne 2), si c'est le cas on ajoute à la variable *condition*, qui représente les conditions de l'URL à générer, la chaîne « live=true ». Cette condition sera analysée par la servlet et les traitements nécessaires seront réalisés.

JQuery a été utilisé afin d'améliorer l'ergonomie de la page, par exemple lorsque l'on clique sur un champs où il faut entrer une date, un calendrier apparaît avec la possibilité de choisir le jour et l'heure. JQuery permet également d'afficher une « sous-fenêtre » pour un formulaire plutôt que de changer de page pour le faire apparaître, ou de la surcharger en le laissant affiché.

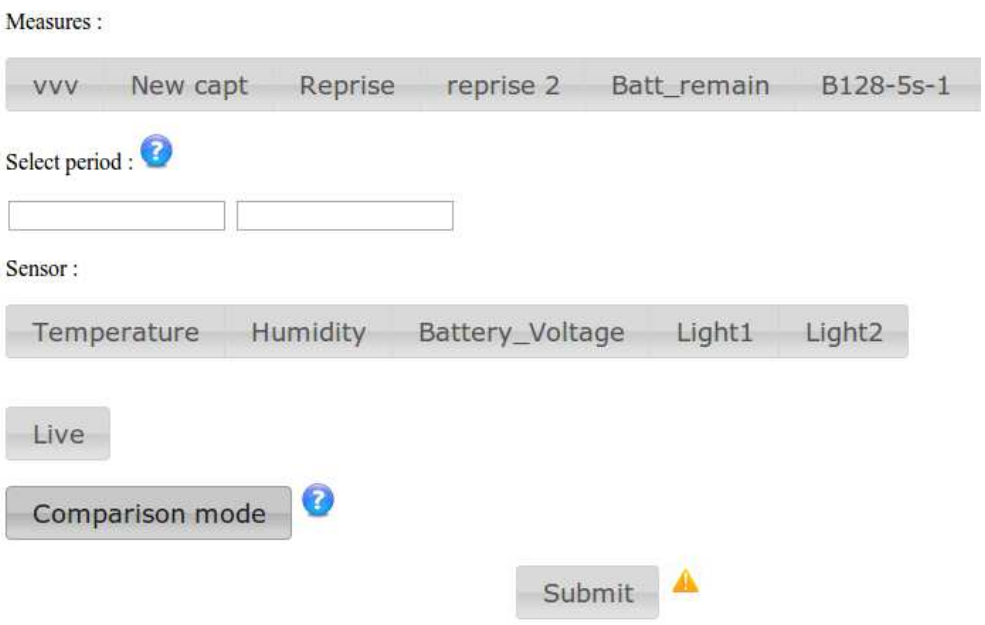
La page de configuration est susceptible de ressembler à la figure 5.2

Comme décrit précédemment, la première ligne présente les différentes mesures réalisées, et la possibilité de lesœ sélectionner, ensuite les dates puis les capteurs, le bouton « live » permet d'activer ou non les données arrivant en live (si le réseau de capteur est toujours connecté). Concernant le bouton « comparison mode », son utilité sera détaillée page 27.

5.2 Second cycle : Les cartes

Suite à différents tests et expériences avec les capteurs et les graphiques, nous nous sommes rapidement rendu compte qu'il n'était pas facile de localiser dans l'espace les capteurs juste avec les courbes et leurs noms. C'est pourquoi nous avons

FIGURE 5.2 – Aperçu de la page de configuration pour les graphiques



décidé de mettre en place une carte représentant les capteurs sur un plan.

La problématique s'est surtout posée au niveau de la conception, quelle technologie utiliser pour générer une carte pratique pour l'utilisateur ? Nous avons d'abord pensé à générer une image sur le serveur puis la transmettre à l'utilisateur. Or, plusieurs limitations sont rapidement apparues, premièrement cette méthode ne permet aucune interactivité, c'est à dire qu'une fois l'image générée, il faut en générer une autre si les données changent ou si l'on veut déplacer un capteur. Deuxièmement, la génération d'image est très lourde pour le serveur, donc le rendu est lent et risque de surcharger l'utilisation des ressources. Et troisièmement, une image n'est pas modifiable, ce qui implique un transfert de toute l'image à chaque modification de paramètre, soit une charge réseau importante.

En conséquence de ces contraintes, la solution restante est de transférer uniquement les données et de générer la carte côté utilisateur, sur le navigateur. Trois technologies répandues étaient disponibles :

1. Applet Java
2. Flash
3. Canvas HTML5 + Javascript

Concernant le premier, nous connaissons Java, ce qui aurait pu être un avantage. Mais les Applet Java nécessitent généralement beaucoup de ressources, et sont connus pour leur manque de fiabilité en matière de sécurité. Concernant Flash, il a l'avantage d'être rapide et très utilisé, mais il est rarement compatible avec les navigateurs sur mobiles. Par ailleurs, c'est un logiciel métier qui nécessite de

l'expérience.

Notre choix s'est donc porté vers la troisième technologie, elle a l'avantage d'avoir déjà été utilisée dans l'application (JavaScript), d'être compatible avec la plupart des navigateurs et d'être relativement rapide. Le *canvas* est une balise HTML apparue dans sa version 5. Au départ c'est un simple espace de pixels transparents, mais couplé à Javascript c'est un outil très puissant. Il peut aller du simple tracé de courbes aux animations, voire aux jeux vidéos. Les *canvas* fonctionnent avec des *frameworks* Javascript, il en existe des dizaines, généralement spécialisés vers des fonctionnalités aussi varié que du graphique à l'interaction matérielle. Nous n'en avions besoin que de certaines assez basiques, comme dessiner des formes et pouvoir interagir avec. Notre choix s'est donc porté vers un framework assez léger, permettant une plus grande rapidité : KineticJS [5].

La phase de codage a commencé une fois la prise en main du framework faite et un approfondissement du rendu final souhaité. Le but étant d'obtenir un *canvas* avec un fond donné, la possibilité de déplacer les capteurs ainsi qu'une représentation de la donnée de température pour chacun. Pour cette représentation nous avons choisi un dégradé de couleur autour des capteurs du rouge au bleu, le rouge étant le plus chaud et le bleu le plus froid. Après discussion, les bornes de froid et chaud sont définies par les températures minimum et maximum de l'ensemble des capteurs à un temps donné plutôt que des valeurs définies. Chaque méthode a ses avantages et ses inconvénients, mais l'objectif de l'application étant un outil de comparaison, des variations de couleurs relatives à l'expérience sont plus pertinentes que des bornes fixes.

Pour réaliser ce dégradé, nous avons utilisé les spécifications de la représentation RGB, c'est à dire que la couleur est sur six octets, les deux premiers concernent le rouge, les deux suivant le vert et les deux derniers le bleu. Pour un dégradé du rouge au bleu, le vert est toujours à « 00 ». Concernant le bleu et le rouge, leur variation vont de 0 à 255, le minimum étant bleu, sa représentation RGB sera 0000FF [6], et inversement pour le rouge : FF0000. Concernant les valeurs intermédiaires, elles sont calculées en fonction de l'écart entre le rouge et le bleu. Par exemple pour le rouge, cette formule a été utilisée : $255 - (maxTmp - forme.temperature) \cdot cste$

`maxTmp` étant la valeur maximum au temps donné, `forme.temperature` la température du capteur concerné, la constante `cste` valant $\frac{255}{(maxTmp - minTmp)}$. De cette manière, nous obtenons une valeur comprise entre 0 et 255 en adéquation avec le minimum et le maximum. Par exemple, une température proche de celle du maximum donnera une valeur au rouge proche de 255 et proche de 0 pour le bleu. Un algorithme a dû être mit en place pour vérifier quelques cas particuliers :

— Un capteur initialement à une valeur intermédiaire devient un maxima

5. `http://kineticjs.com`
6. FF étant 255 en hexadecimal

— Un des maximas change de valeur mais ne devient pas intermédiaire
— Une valeur intermédiaire change sans dépasser un maxima

Dans les deux premiers cas, le changement implique généralement un changement de l'écart et implique un nouveau calcul de la couleur de chaque capteur. Le deuxième cas existe car si le maximum devient légèrement inférieur à l'ancien maximum alors il faut tout de même recalculer toutes les couleurs, ce qui mène à une surveillance de changement d'état des maximas et pas uniquement un dépassement des bornes. Les calculs de changement de couleur auraient très bien pu être effectué à chaque nouvelle donnée, mais pour des raison d'optimisation il est plus efficace d'effectuer une surveillance du minimum et maximum.

Nous avons également fait le choix d'ajouter la possibilité de se déplacer dans le temps sur la carte sans avoir à changer de page. Pour cette fonctionnalité nous avons implémenté un curseur de temps en JQuery. Le curseur va de la première date sur la plage de donnée reçues à la dernière. Grâce à JQuery il est possible d'effectuer une actions à chaque déplacement du curseur et donc d'afficher sur la carte les données à jour pour la date voulue. Or tous les capteurs ne transmettent pas une donnée en même temps, il a donc fallu mettre en place un algorithme pour chercher les points les plus proches de la date voulue. Concernant cet algorithme, bien qu'une recherche dichotomique aurait été intéressante, elle n'a pas pu peut être mise en place car les données d'un capteur ne sont pas reçue à intervalles réguliers. Nous avons donc opté pour une recherche séquentielle dont voici le code :

```
1  function auPlusProche(id, date, liste){
2  var distTemps, obj;
3  for(var j = 0; j<liste.length; j++){
4          if(liste[j].mote == id){
5                  ecartTmp = Math.abs(date-liste[j].timestamp)
6                  if(typeof distTemps == 'undefined'
7                      || ecartTmp<distTemps){
8                          distTemps=ecartTmp;
9                          obj = liste[j];
10                 }
11                 if(ecartTmp>distTemps){
12                         break;
13                 }
14         }
15 }
16         return obj
17 }
```

Afin d'optimiser tout de même ce parcours séquentiel, nous avons mis en place

une variable `distTemps` stockant la « distance » en temps de la donnée par rapport à la date voulue. Tant que `distTemps` devient plus petite d'une itération à une autre on remplace l'objet `obj`, qui représente la donnée la plus proche (ligne 7 à 9). Enfin, si `distTemps` devient plus grande, c'est que l'on s'éloigne de la date, et donc on s'arrête (ligne 11-12) puis on retourne l'objet (ligne 16).

Nous pouvons voir sur la figure 5.3 l'exemple d'un rendu :

FIGURE 5.3 – Aperçu du rendu d'une carte

Sur le haut, le curseur permettant de se déplacer dans le temps, en dessous une image générée par Sweet Home 3D [7] avec la disposition des capteurs ainsi que les variations de couleurs correspondant aux variations de température.

5.3 Troisième cycle : La comparaison

L'application permet de créer différentes mesures pour une expérience, or, jusqu'à présent nous ne pouvons analyser les résultats des mesures uniquement les uns après les autres. Il manquait la possibilité de pouvoir comparer sur une même page les données, que ce soit sur les graphiques ou sur les cartes.

7. `www.sweethome3d.com`

5.3.1 Graphiques

Nous avons d'abord implémenté l'option de comparaison aux graphiques. Initialement, si plusieurs mesures étaient sélectionnées sur la page de configuration, les données s'affichaient à la suite sur le graphique, comme si c'était une seule et même mesure. Le but est de pouvoir superposer les courbes avec comme origine de temps le début de la mesure.

Sur la page JSP du graphique rien n'a été à modifier, les données étant à traiter de la même manière. Le travail a surtout été présent sur la servlet et sur la page de configuration. Sur cette dernière il a fallut ajouter une option « comparaison » sous forme de bouton. Ce bouton laisse apparaître deux nouveaux champs de date, permettant de saisir une date relative, c'est à dire un nombre d'heures et minutes depuis le début de l'expérience et non une date fixe. La sélection de ce bouton entraîne la transmission d'un nouveau paramètre à la servlet : `mode=compare`, et donc un nouveau traitement par celle-ci.

Toute la différence entre le mode « normal » et le mode « comparaison » se trouve dans la servlet qui effectue un traitement spécial sur les données. En effet, toutes les données sont récupérées via la même requête SQL que pour le mode « normal ». Mais par la suite, pour chaque donnée de chaque mesure la date de la mesure se voit changée par : $dateDeLaMesure - dateDebutMesure$. Ce calcul permet d'obtenir un temps par rapport au début de la mesure. De plus l'identifiant de chaque capteur est concaténé avec le nom de l'expérience, ce qui permet de différencier les mesures dans la légende du graphique.

5.3.2 Cartes

L'implémentation du mode de comparaison, lui, s'est effectué sur les trois parties : La page JSP, la servlet, et la page de configuration.

Tout d'abord, sur la page de configuration, les modifications ont été minimes, c'est à dire remplacer les formulaires de date par des formulaires avec des temps relatifs, tout comme pour les graphiques.

Ensuite, concernant la servlet, une nouvelle donnée devait être transférée en plus de celles transmises dans le mode « normal ». Cette donnée étant une liste contenant la première donnée de chaque mesure, elle est utilisée par Javascript pour le calcul de temps par rapport au début de l'expérience. Contrairement aux graphiques, le calcul du temps par rapport au début de l'expérience se fait en Javascript, ce choix a été fait pour une raison technique. Car pour la carte, il y a assez peu de chance que **toutes** les données soient utilisées. En effet seules les données balayées par le curseur seront utilisées, donc pour économiser des opérations, le calcul est fait lorsque la donnée est nécessaire.

Enfin du côté JSP, une nouvelle page a été créée qui génère un *canvas* pour

chaque expérience en transmettant la bonne plage de donnée pour chacun d'eux. Le code suivant est chargé de cette tâche :

```
 1 <c:set var="count" value="1"/>
 2 <c:forEach var="i" items="${listGlobal}">
 3         <label for="slider">Date:</label>
 4         <input type="text" class="amount"/>
 5         <div class="slider" id="slider${count}"></div>
 6         <c:set var="lastData" value="${i}" scope="request"/>
 7         <jsp:include page="canvasMapCompare.jsp">
 8         <jsp:param value="false" name="live" />
 9         <jsp:param value="container${count}" name="nameDiv"/>
10         <jsp:param value="${count}" name="count"/>
11         </jsp:include>
12         <c:set var="count" value="${count+1}"/>
13 </c:forEach>
```

Ce code effectue une boucle sur chaque composant de la liste `listGlobal` qui contient toutes les données demandées pour chaque expérience (ligne 2 à 13). Pour chaque itération, un champs de saisie pour la date souhaité est créé (ligne 3 et 4), ainsi qu'un `slider` associé à ce champs. La valeur du `slider` est écrite dans le champs de saisie. Ensuite, les données pour l'expérience concernée par l'itération sont mises à disposition pour toutes les pages de la requête (ligne 6). Enfin, ligne 7 à 11, on inclut la page JSP qui contient le code du *canvas*.

La page qui contient le code du *canvas* est légèrement modifiée par rapport à celle du mode « normal ». Comme précisé précédemment le calcul de date se fait sur cette page, les modifications nécessaires ont donc été apportées. De plus un *listener*[8] a été ajouté sur le *slider*. Il a pour rôle de déclencher une méthode mettant à jour chaque carte à chaque fois que le curseur est lâché par la souris sur une date.

Nous avons mis en place un cas particulier pour la carte rattachée au curseur déplacé. Elle se met à jour à chaque déplacement de celui-ci, ce qui donne un effet de « film en accéléré » si l'on déplace de façon linéaire le curseur. Nous n'avons pas pu mettre en place cette animation pour toutes les cartes sur la page pour des raisons de performance. En effet, le nombre d'opérations à effectuer en même temps est trop élevé pour un langage de script si toutes les cartes se mettent à jour en même temps. Nous nous sommes donc limité à la carte, sous le curseur déplacé, les autres étant misent à jour lorsque le curseur est relâché.

8. Partie logicielle en attente d'un événement

5.4 Quatrième cycle : Surveillance

Jusqu'à présent orienté vers l'analyse et la comparaison, notre application doit être la plus universelle possible, et la démonstration d'une application similaire à la nôtre par un entrepreneur extérieur nous a orienté vers de nouvelles possibilités.

Au milieu de la 6ème semaine de stage, le directeur technique de B Eco Manager [9] est venu car intéressé par les capteurs sur lesquels nous travaillons. L'entreprise est spécialisée dans les solutions de maîtrise énergétique et utilise différents capteurs pour optimiser la consommation. Leur application de supervision propose des fonctionnalités reposant sur les mêmes bases que ce que nous avons développé, excepté la surveillance des valeurs.

Nous avons donc décidé d'implémenter une base permettant de gérer une surveillance via des règles particulières définies par l'utilisateur. Par exemple définir une plage de température acceptable pour un capteur. Afin de mettre en place ce système nous avons dû modifier la base de donnée et ajouter une table « Monitoring Rules » qui stocke toutes les règles ajoutées, ainsi qu'une table « Alerts » chargée de stocker toutes les alertes ayant été relevées. Nous avons dû ajouter les *JavaBeans* nécessaires pour correspondre avec la base, c'est à dire un *Javabean* pour la classe « Alert », et un pour la classe « Monitoring Rules ». Pour que ces deux *JavaBeans* puissent interagir avec la base de données, ils ont besoin d'un DAO, donc pour chacun, une classe héritant du DAO a dû être adaptée.

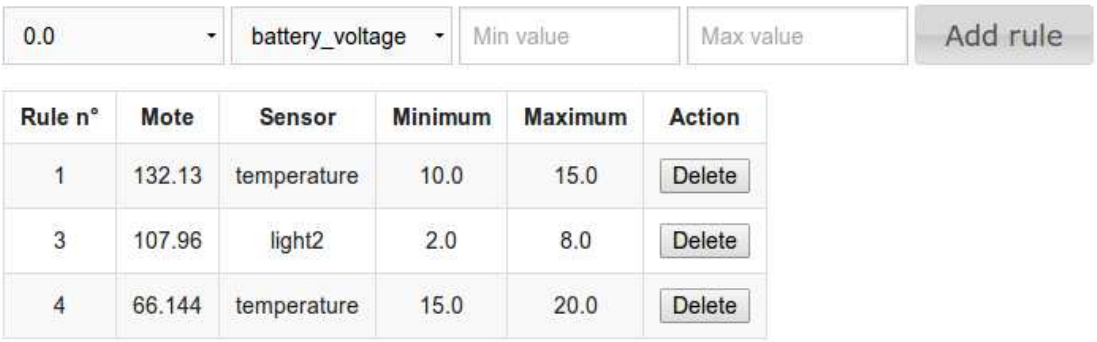
Pour réaliser l'interface graphique de ces règles, nous avons ajouté un onglet à une page d'administration déjà existante. Cet onglet a pour objectif de présenter les règles déjà entrées, et de laisser la possibilité d'en ajouter de nouvelles. Pour afficher les règles déjà existantes, il faut les transmettre via la servlet. Une première modification a été faite dans la servlet de la page d'administration. Elle consiste à récupérer toutes les données de la table « Monitoring Rules », les transformer au format JSON, et les transmettre. Sur cette dernière, on génère un tableau en réalisant une boucle sur chaque règle. Concernant la création de règle, quatre champs sont nécessaires, la capture de la figure 5.4 montre l'interface finale.

Sur le haut de la figure, on remarque les quatre champs de saisie, les deux premiers sont des listes déroulante laissant le choix entre tous les capteurs du réseaux pour la première, et le choix de tous les types de *sensor* pour la seconde (temperature, humidité, ...). Ces listes sont générées par JSTL via des données transmises par la servlet. Les deux champs suivants sont les bornes dans lesquelles des valeurs sont autorisées. Une fois la règle créée, si un capteur envoie une valeur en dehors de ces bornes un événement est déclenché.

Nous avons choisi de créer une première mise en situation pour la surveillance, c'est à dire l'envoi de mail lorsqu'un capteur sort des bornes. Cette implémentation

9. `www.ecomanager.fr`

FIGURE 5.4 – Aperçu de la page d'administration des règles de supervision



se situe entièrement dans le code Java, elle s'effectue en deux parties :

- La première est de mettre en place une classe capable d'envoyer un mail avec un sujet et un texte donné. Des objets Java sont déjà prévus à cet effet dans `Java.mail`. Concernant le serveur SMTP [10], serveur qui permet l'envoi de mail, nous avons choisi de ne pas l'héberger nous même, car cela rendrait la tâche plus ardue pour un futur utilisateur de l'application, nous nous somme donc basé sur le serveur de Yahoo Mail.
- La deuxième chose a faire est de mettre en place une méthode générant le texte pour le mail en fonction du capteur concerné par la règle. Enfin il faut placer une méthode vérifiant les valeurs reçues par rapport au règles, qui déclenchera l'envoi du mail et stockage de l'alerte dans la table « Alert ».

10. Simple Mail Transfer Protocol

Quatrième partie

Conclusion

Le monde de la recherche ne peut se baser sur du « déjà vu », le rôle du développeur dans cet environnement est aussi de savoir proposer de nouvelles choses et surtout de laisser son application la plus ouverte possible aux nouvelles idées.
Ce stage a été l'opportunité de mettre à profit les compétences de développement acquises à l'IUT, d'apprendre de nouvelles technologies comme les réseaux de capteurs, le JEE, le Javascript et JQuery. Mais également de développer une application vouée à être reprise par d'autres développeurs.
Le travail rendu permet, comme souhaité, d'analyser et de comparer des données provenant de réseaux de capteur, et ce de différentes manières.
L'implémentation de nouvelles fonctionnalités a été pensé et pourra se faire sans interférer avec le travail déjà réalisé, ce qui facilite l'évolution de l'application.
Ce stage a été l'occasion de confirmer des ambitions déjà présente dans le monde de l'informatique, voire dans la recherche en informatique.